BLM Integrator / Digitizer

FPGA Logic and Functions

I. Introduction

II. Integrator Sequence Control

III. Digitizer Readout and Control

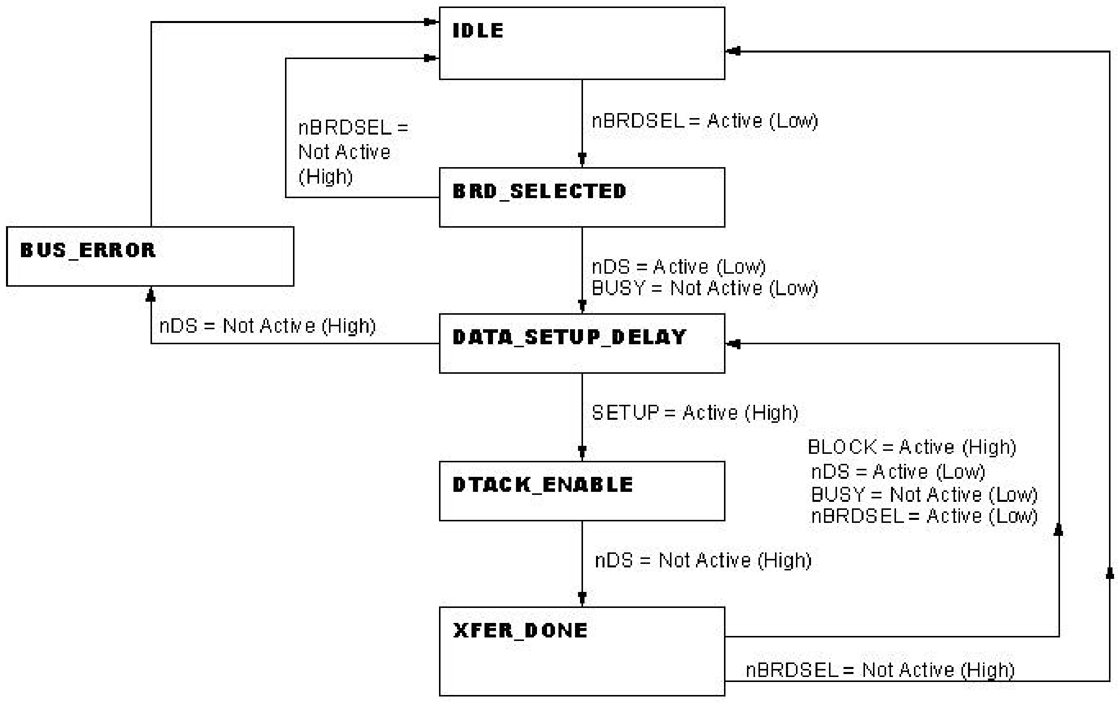
IV. VME Bus Interface

The data transfer modes supported are A24D16 modes.

i.	Standard Supervisory Block Transfer,	AM[5..0] = 3F
ii.	Standard Supervisory program access,	AM[5..0] = 3E
iii.	Standard Supervisory data access,	AM[5..0] = 3D
iv.	Standard Non-privileged Block Transfer,	AM[5..0] = 3B
v.	Standard Non-privileged program access,	AM[5..0] = 3A
vi.	Standard Non-privileged data access,	AM[5..0] = 39

Program access is treated exactly the same as data access. Supervisory modes are treated exactly the same as non-privileged modes. Block transfers are supported.

The logic has mainly two parts. The first is a state machine that handles the handshaking and timing for the VME data transfer. Figure IV.2.1 is a representation of the state transitions. The second is an address decoder that produces the necessary latches and chip selects to Read or Write the registers implemented in the FPGA or the SRAM memory. The address decoder portion also generates the necessary control signals for the VME data bus transceivers.



INPUTS

```
nBRDSEL = !( (nIACK & nLWORD & AM5 & AM4 & AM3 &  AM1 & !AM0 & !nAS & !nAMATCH)
              # (nIACK & nLWORD & AM5 & AM4 & AM3 & !AM1 &  AM0 & !nAS & !nAMATCH)
              # (nIACK & nLWORD & AM5 & AM4 & AM3 &  AM1 &  AM0 & !nAS & nAMATCH));
BLOCK   = !nBRDSEL & AM1 & AM0; % Block transfer indication %
nDS     = nDS1 & nDS0;
BUSY    = Indication that the VME bus does not yet have access to the SRAM .
SETUP   = Active when the data setup delay count reaches 5.
          Total delay is 6 time logic clock period
```

OUTPUTS

```
DTACK      = DTACK_ENABLE;
BERR       = BUS_ERROR;
nWR_GATE = !(DATA_SETUP_DELAY);
nRD_GATE  = !(DATA_SETUP_DELAY # DTACK_ENABLE);
```

Figure IV.2.1 VME Interface State Transition Diagram

In Figure IV.2.2 gives the timing diagram for a VME transaction. This timing is general for either Reads or Writes. The state of the VME WRITE/ line will determine how the RD_GATE/ and WR_GATE/ signals are used. This VME bus interface is particularly slow. The design does allow a lot of setup and hold time for the data lines. All of the incoming VME bus signals are synchronized to the local logic clock (40ns). This also adds to the total transfer time, but is essential in order to avoid crashing the state machine, as it is written. Extra time is also taken up by the SRAM memory access control (V_Request, V_Busy, V_Access). The memory access control logic is discussed in its own section.

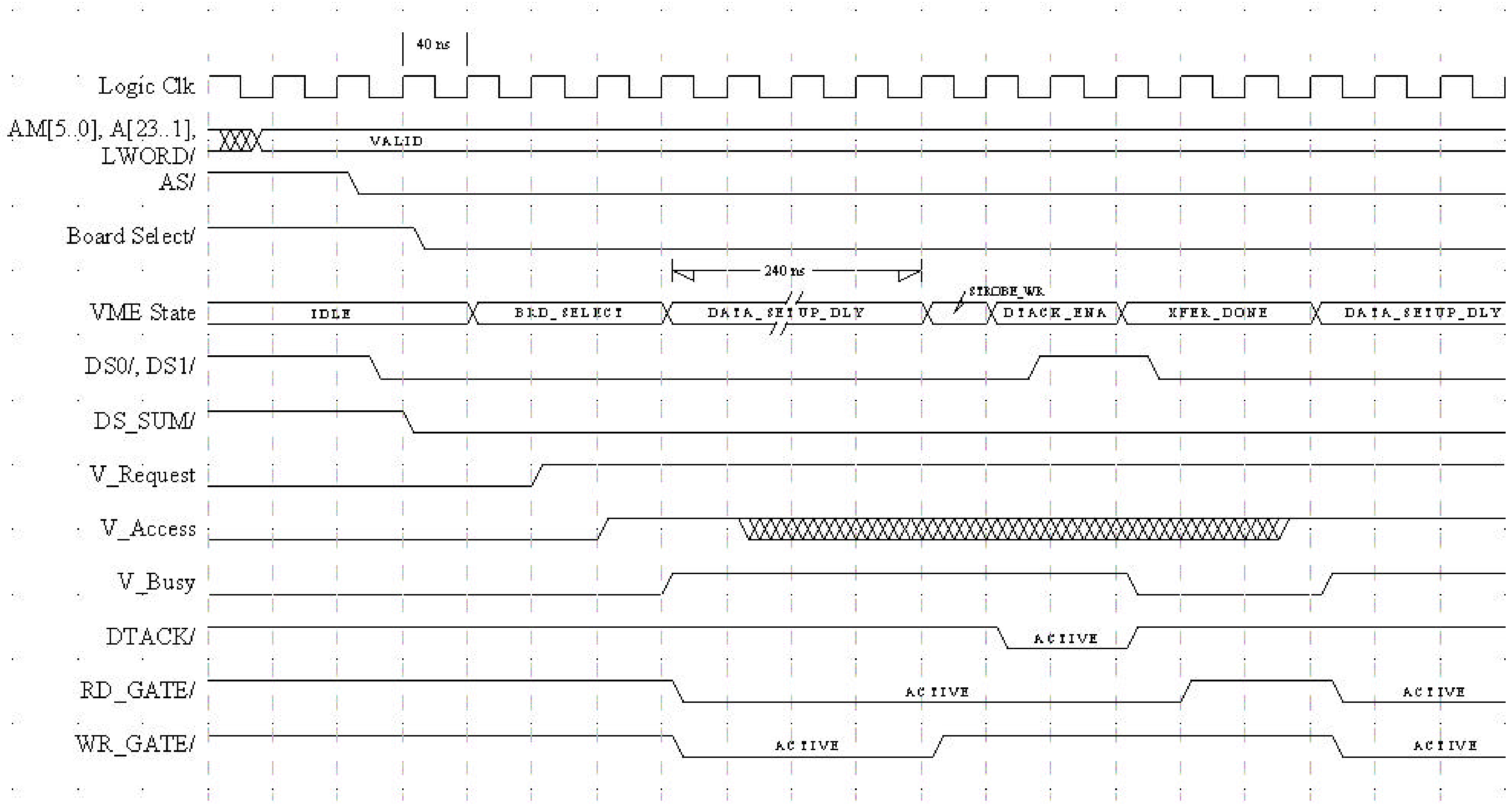


Figure IV.2.2 VME Interface Timing Diagram

The 24 VME address lines are used as indicated in Figure IV.2.3. The most significant 5 bits (A23 to A19) are compared to DIP switches on the module to determine if the particular board is being addressed. Bits A18 and A17 determine which of the 4 channels on the board is involved. The final 16 bits are used to address memory and registers for the selected channel.

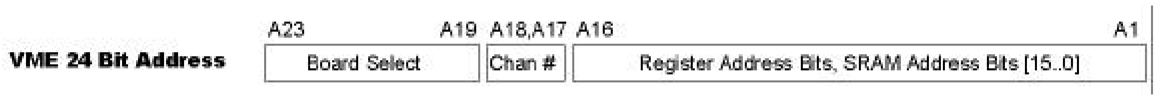


Figure IV.2.3 VME Address Bits

The memory map ***(as of Sept. 2004)*** is given in Table IV.2.1. Note that all 65,536 locations in the SRAM memory for each channel can only be accessed via the circular buffer interface. The most upper 16 addresses (0xFFF0 to 0xFFFF) had to be set aside as control and status registers. This still leaves 65,520 addresses in each channels SRAM that can be accessed individually.

Table IV.2.1 Integrator / Digitizer Memory Map

Description	**Address [15..0]**	**R/W**
SRAM Circular Buffer	0xFFFF	R only
Command Register	0xFFFE	R/W
Status Register	0xFFFD	R only
Integrator Values in SRAM	0x0000 to 0xFFEF	R only

The Altera AHDL logic that implements this VME interface is listed in Appendix A.

SRAM Memory Access Control

V.1 The Circular Buffers

Each of the four integrator channels have 64K x 16 bits of SRAM memory on the board. Every 21 us the digitized value from each channel is written into its memory. The memory is used as four circular buffers. Every time a value is written into the memory the Input Pointer is advanced. When the Input Pointer reaches the final memory location in the 64K SRAM (65,535), it wraps back around to the memory location 0. Also an Output Pointer is maintained. This pointer points to the oldest value in memory. The Output Pointer is advanced whenever the SRAM circular buffer is read by the VME interface. The Output Pointer also advances whenever the Input Pointer has wrapped all the way back to the Output Pointer and the Input Pointer is advanced.

Since each of the four integrator channels are updated every 21 us it should always be the case that the Input Pointer for every channel is the same value. The only process in the current application intended to be writing values into the circular buffers is the storing of integrator values every 21 us.

V.2 Processes That Use the SRAM Memory

There are three different processes implemented in the FPGA logic that Read and/or Write the SRAM memory as indicated in Figure V.2.1. .

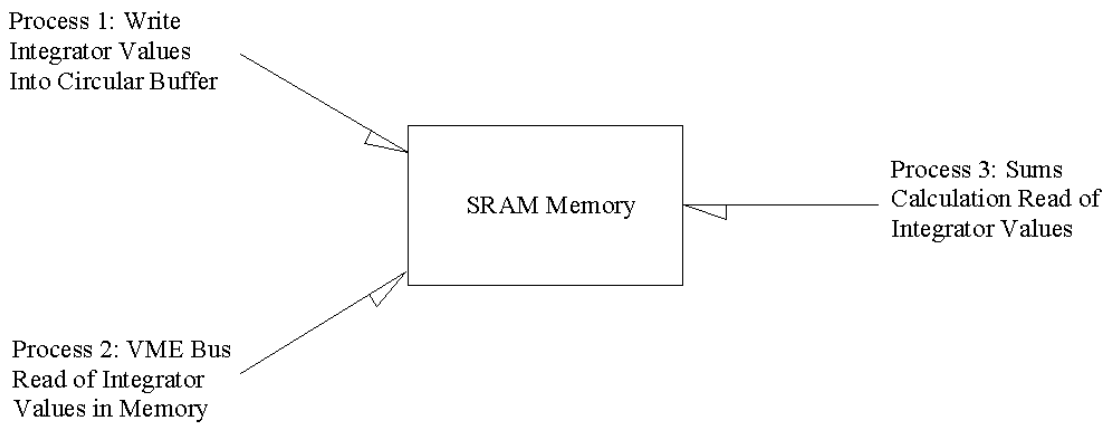

Process 1: Write Integrator Values Into Circular Buffer
SRAM Memory
Process 3: Sums Calculation Read of Integrator Values
Process 2: VME Bus Read of Integrator Values in Memory

Figure V.2.1 Three processes that access memory.

Every 21 us there is a need for FPGA 1 to write the four integrator values into memory. The Sums are computed every 21 us in FPGA 2. The memory is Read by this process. Integrator values can be Read from the memory via the VME interface.

Note that these are not three different subroutines of a program running in a single processor, but these are three logical function blocks made up of their own independent set of logic gates, running simultaneously.

Through the VME interface, each memory location below 0xFFF0 (65,520) is individually accessible. For applications like the Booster that runs for no more than 40 ms and then starts again this is plenty of memory. For applications like the Tevatron where integrator values are updated continuously, the full record (65,536 measurements) can be read via the circular buffer mapped at address 0xFFFF. Control and status registers are mapped into the addresses between 0xFFF0 and 0xFFFE.

V.3 Memory Access Arbitration

The four 64K x 16 bit SRAM memory chips are all connected with a common 16 bit address and 16 bit data buses, as well as a common Write signal line (WR/) and a common output enable line (OE/). The output enable is used when the memory is being Read. Each memory chip has its own chip select (CS/) line that is used to select the individual memory chip that is meant to respond. Schematically, each of the three processes that access the memory have their own set of these memory control lines.

However the memory control lines of only one process at a time are allow to drive the actual memory control lines connected to the memory chips.

Logic in FPGA 1 implements a simple scheme for arbitrating which process is allowed access to the memory. Each process has two logical signals it sets and clears. These are the Request line and the Busy line. Each process has a single logical input that indicates that it has access to the memory. This is the Access line. The scheme is as follows.

1) Logical processes wanting to Read or Write from the SRAM memory must set their Request line (Rqst) active and wait for their Bus Access line (Access) to become active before beginning their memory access.

2) Before releasing their Request line, their Busy line should be set active. Note that the Request line may remain active until the memory access has been completed. Then the Request line must be released before or simultaneously with the Busy Line going inactive.

3) No other logical process will be granted access to the memory while the process that was granted access, holds its Busy line active.

4) If a higher priority process sets its Request line active while a lower priority process is Busy, the Bus Access line for the lower priority process will go inactive indicating that the lower priority process *should* suspend its memory access and set its Busy line inactive. The lower priority process should keep its Request line active if it had not completed its memory access.

5) If a lower priority process sets its Request line active while a higher priority process is Busy, the status of the higher priority process' Bus Access line will not change. The lower priority process will not see its Bus Access line go active until the higher priority process has released its Busy line.

6) There are no registers in which priorities are set as there are in some microprocessors. The priority is hard coded into the state machine that implements the arbitration.

To describe this another way, a typical sequence of events for a processes memory access would be

1) Set Request line active.

2) Wait for Bus Access line to go active.

3) Set Busy line active and do memory Read and/or Write.

4) If while busy accessing the memory the Bus Access line goes inactive suspend further memory accesses, set the Busy line inactive, and leave the Request line active if the memory access was not completed.

5) When the Bus Access Line goes active once again set the Busy Line active again and continue the memory access.

6) Once the memory access is complete set both the Busy Line and the Request Line inactive.

Note that a processes memory access is considered to be any number of Reads and/or Writes and the arbitration scheme does not force immediate pre-emption of a lower priority process' memory access. A process is expected to suspend its memory access at an appropriate point for that process.

V. Analog Output DAC Loading

stray stuff ==

Control Bus Logic

1. A signal from the Integrator control logic will initiate the following sequence
   a) The Control Bus FPGA will set its Bus Request bit signal and wait for its Bus Access bit to go active.
   b) The Bus Busy bit signal will be set active.
   c) A signal to the sums logic is generated to start the computation of new sums.
   d) When the new sums have been computed the values are latched into registers and the new sums are compared to their associated thresholds.
   e) If any of the twelve sums (4 channels x 3 different length sums) or any of the 4 immediate integrator values are over threshold the appropriate abort bit is set.
   f) When the computation of the sums is complete the Bus Request and Bus Busy bit will be set inactive.

2. The registers holding the value of the sums will be double buffered by following the registers with latches that will be latched by the Control Bus backplane signals Fast_LATCH(0), Slow_LATCH(1), and Vslow_LATCH(2).

3. The signals AbortCS[4..2] are compared to the jumper selected address on the module. When there is a match the registers holding the Abort status bits are latched and the signals AbortCS[1..0] sequence through the four sets of four Abort status bits, putting their values on the Control Bus lines ABORT[3..0]..

APPENDIX A

Intg_vme.tdf: Altera AHDL file that implements the VME interface for the Integrator Digitizer board.

```
CONSTANT SETUP_DELAY_COUNT = B"0101";

SUBDESIGN Intg_vme
(
        clk, reset/, A[18..1], V_Access : INPUT;
        AM[5..0], IACK/, LWORD/, AS/, AMATCH/ : INPUT;
        DS0/, DS1/, WRITE/  : INPUT;

        vmePTR[15..0], V_Busy, V_Rqst : OUTPUT;
        LEAB/, CEAB/, OEAB/, OEBA/, DTACK, BERR : OUTPUT;

        RD_CirBuf1,   RD_CirBuf2,   RD_CirBuf3,   RD_CirBuf4 : OUTPUT;
        RD_Command, WR_Command, RD_Status  : OUTPUT;
        vme_MEM_CS/1, vme_MEM_CS/2, vme_MEM_CS/3, vme_MEM_CS/4 : OUTPUT;

        RD_SRAM/, WR_SRAM/, vmeSRAM_ACCESS, nBRDSEL : OUTPUT;

        RD_GATE/, WR_GATE/, STATE[2..0], nDS_SUM : OUTPUT; %Simulation Testpoints%
)

VARIABLE
    nBRDSEL                   : LCELL;
        BLOCK                 : LCELL;

        sync_DS[1..0]   : DFF;
        nDS                           : LCELL;
        sync_write      : DFF;
        vmeWRITE/       : LCELL;
        sync_AS/        : DFF;
        vmeAS/          : LCELL;

        VME_SM   : MACHINE
                              OF BITS (STATE[2..0])
                              WITH STATES (
                                      IDLE           =B"000",
                                      BRD_SELECTED   =B"001",
                                      DATA_SETUP_DLY =B"101",
                                      STROBE_WR      =B"011",
                                      DTACKEN        =B"100",
                                      XFER_DONE      =B"010",
                                      BUS_ERROR      =B"110",
                                      XSTATE1        =B"111" );
        CNT[3..0]      : DFFE;
        SETUP          : LCELL;
        BUSY        : LCELL;
        ALATCH[18..1] : DFFE;
        CHAN[1..0]    : NODE;

        r1  : DFF; % Helps synchronize combinatorial logic for the RD_GATE/ signal %


BEGIN
%=================================================================================%
% VME Handshaking Logic                                                           %

        % INPUTS ==================================================================%
        % Synchronize the VME signals to the local logic clock %
        sync_DS[].clk  = clk;
        sync_DS[].prn  = reset/;
        sync_DS1.d = DS1/;
        sync_DS0.d = DS0/;

        nDS   = sync_DS1.q & sync_DS0.q;

        nDS_SUM = nDS; %nDS_SUM is just a testpoint for simulation%

        sync_write.clk  = clk;
        sync_write.prn  = reset/;
        sync_write.d = WRITE/;    %Note: The VME WRITE/ line will be valid for at least%
                                  %      10ns after the DS/ lines have terminated the  %
                                  %      transfer.                                     %
        vmeWRITE/ = sync_write.q;

        sync_AS/.clk  = clk;
        sync_AS/.prn  = reset/;
        sync_AS/.d = AS/;

        vmeAS/ = sync_AS/.q;

        % STANDARD ADDRESS MODIFIER DECODE ============================================%
        nBRDSEL = !(
                          (IACK/ & LWORD/ & AM5 & AM4 & AM3 &  AM1 & !AM0 & !vmeAS/ & !AMATCH/)
                        # (IACK/ & LWORD/ & AM5 & AM4 & AM3 & !AM1 &  AM0 & !vmeAS/ & !AMATCH/)
                        # (IACK/ & LWORD/ & AM5 & AM4 & AM3 &  AM1 &  AM0 & !vmeAS/ & !AMATCH/));

        BLOCK = !nBRDSEL & AM1 & AM0; % Block transfer indication %

    % Delay Counter to guarantee a sufficient data setup time ====================%
        CNT[].clk = clk;
    CNT[].clrn = !(IDLE#XFER_DONE);
        CNT[].ena  = !SETUP;

    IF DATA_SETUP_DLY THEN
       CNT[].d = CNT[].q + 1;
    ELSE
```

```
      CNT[].d = CNT[].q;
   END IF;

   IF (CNT[].q == SETUP_DELAY_COUNT) THEN
      SETUP = VCC;
   ELSE
      SETUP = GND;
   END IF;

       % Memory bus arbitration variables ==============================================%
       %   refer to "mem_arbitration.tdf" for an explanation of the arbitration protocol%

       BUSY = !V_Access;  % This says Hold-off, the memory is busy, VME doe not have access%
       V_Rqst = (!nBRDSEL & !nDS);                 % This is the VME's request for the bus %
       V_Busy = (DATA_SETUP_DLY # STROBE_WR # DTACKEN);   % This line holds off all others %

       % OUTPUTS ======================================================================%
       DTACK    = DTACKEN;
       BERR     = BUS_ERROR;

       WR_GATE/ = !DATA_SETUP_DLY;

       r1.clk   = clk;                                        % This bit of logic holds   %
       r1.prn   = reset/;                                     % RD_GATE/ low and smooths  %
       r1.d     = !(DATA_SETUP_DLY # STROBE_WR # DTACKEN);       % over possible glitches due%
       RD_GATE/ = !(DATA_SETUP_DLY # STROBE_WR # DTACKEN) & r1.q;% to combinatorial logic    %

% VME STATE MACHINE ============================================================%
       VME_SM.clk = clk;
       VME_SM.reset = !reset/;
TABLE
reset/,nBRDSEL, BLOCK, nDS, SETUP, BUSY, VME_SM         => VME_SM;
 % Reset %
   0  ,   X   ,   X  ,  X ,   X  ,  X  , IDLE           => IDLE;
 % IDLE  %
   1  ,   1   ,   X  ,  X ,   X  ,  X  , IDLE           => IDLE;
   1  ,   0   ,   X  ,  X ,   X  ,  X  , IDLE           => BRD_SELECTED;           % got a board select %
 % BOARD SELECTED %
   1  ,   1   ,   X  ,  X ,   X  ,  X  , BRD_SELECTED   => IDLE;                % lost the board select %
   1  ,   0   ,   X  ,  1 ,   X  ,  X  , BRD_SELECTED   => BRD_SELECTED;        % wait for DS to go low %
   1  ,   0   ,   X  ,  0 ,   X  ,  1  , BRD_SELECTED   => BRD_SELECTED;            % HOLDOFF IS ACTIVE %
   1  ,   0   ,   X  ,  0 ,   X  ,  0  , BRD_SELECTED   => DATA_SETUP_DLY;                % received DS %
 % DATA_SETUP_DLY %
   1  ,   X   ,   X  ,  1 ,   X  ,  X  , DATA_SETUP_DLY => BUS_ERROR; % Lost the data strobe -> BUS ERR %
   1  ,   X   ,   X  ,  0 ,   0  ,  X  , DATA_SETUP_DLY => DATA_SETUP_DLY;             % wait for 200ns %
   1  ,   X   ,   X  ,  0 ,   1  ,  X  , DATA_SETUP_DLY => STROBE_WR;                     % setup over %
 % STROBE_WR %
   X  ,   X   ,   X  ,  X ,   X  ,  X  , STROBE_WR      => DTACKEN; % Ensures one clock data hold time %
 % DTACK ENABLE %
   1  ,   X   ,   X  ,  0 ,   X  ,  X  , DTACKEN        => DTACKEN;        % wait for DS to go inactive %
   1  ,   X   ,   X  ,  1 ,   X  ,  X  , DTACKEN        => XFER_DONE;           % transfer is completed %
 % TRANSFER COMPLETED %
   1  ,   0   ,   X  ,  1 ,   X  ,  X  , XFER_DONE      => XFER_DONE;   % wait for nDS or AS/ to change %
   1  ,   0   ,   0  ,  X ,   X  ,  X  , XFER_DONE      => XFER_DONE;   % wait for nDS or AS/ to change %
   1  ,   0   ,   1  ,  0 ,   X  ,  1  , XFER_DONE      => XFER_DONE;               % HOLDOFF IS ACTIVE %
   1  ,   0   ,   1  ,  0 ,   X  ,  0  , XFER_DONE      => DATA_SETUP_DLY;% continue the block transfer %
   1  ,   1   ,   X  ,  X ,   X  ,  X  , XFER_DONE      => IDLE;  % all done -> IDLE %
 % BUS ERROR %
   1  ,   X   ,   X  ,  X ,   X  ,  X  , BUS_ERROR      => IDLE;  % Go Idle %
 % ILLEGAL STATES %
   1  ,   X   ,   X  ,  X ,   X  ,  X  , XSTATE1        => IDLE;  % Go Idle %
END TABLE; %==============================================================================%

%================================================================================%
% VME Address Decoding Logic                                                     %

       ALATCH[].clk = clk;               %This latch guarantees the address is held %
       ALATCH[].clrn = reset/;           %for as long as we may need it.            %
       ALATCH[].d   = A[18..1];
       ALATCH[].ena = RD_GATE/;

       CHAN[1..0]    = ALATCH[18..17].q;
       vmePTR[15..0] = ALATCH[16..1].q;  %Address pointer out to SRAM Memory%

       RD_CirBuf1 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 0) & (ALATCH[16..1].q == H"FFFF");
       RD_CirBuf2 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 1) & (ALATCH[16..1].q == H"FFFF");
       RD_CirBuf3 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 2) & (ALATCH[16..1].q == H"FFFF");
       RD_CirBuf4 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 3) & (ALATCH[16..1].q == H"FFFF");

       RD_Command  = !RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q == H"FFFE");
       WR_Command  = STROBE_WR & (vmeWRITE/ == GND) & (ALATCH[16..1].q == H"FFFE");

       RD_Status   = !RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q == H"FFFD");

       vmeSRAM_ACCESS = (!RD_GATE/ & (A[16..1] < H"FFF0"));

       RD_SRAM/ = !(!RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q < H"FFF0"));

       WR_SRAM/ = !(!WR_GATE/ & (vmeWRITE/ == GND) & (ALATCH[16..1].q < H"FFF0"));

       vme_MEM_CS/1 = !(!RD_GATE/ & (CHAN[] == 0) & (ALATCH[16..1].q < H"FFF0"));
       vme_MEM_CS/2 = !(!RD_GATE/ & (CHAN[] == 1) & (ALATCH[16..1].q < H"FFF0"));
       vme_MEM_CS/3 = !(!RD_GATE/ & (CHAN[] == 2) & (ALATCH[16..1].q < H"FFF0"));
       vme_MEM_CS/4 = !(!RD_GATE/ & (CHAN[] == 3) & (ALATCH[16..1].q < H"FFF0"));

       % Logic for the bus transceiver control lines ==========================================%
       LEAB/ = !(!RD_GATE/ & (vmeWRITE/ == VCC));  % Doing a vme read %
       CEAB/ = LEAB/;
       OEAB/ = LEAB/;

       OEBA/ = !(!RD_GATE/ & (vmeWRITE/ == GND));  % Doing a vme write %
END;
```